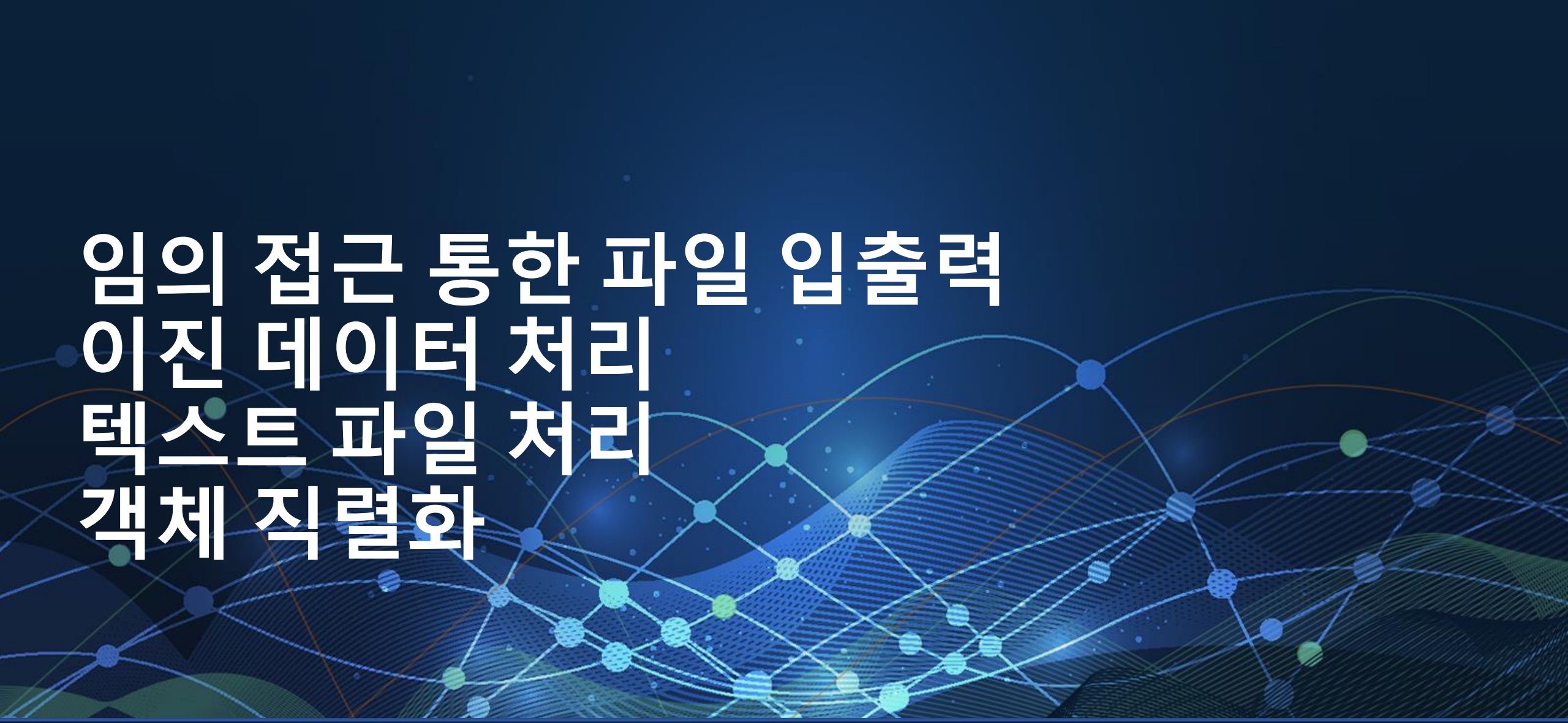
C# 프로그래밍

- 11 주차 (1강)

SEJONG UNIVERSITY

임의 접근 통한 파일 입출력
이진 데이터 처리
텍스트 파일 처리
객체 직렬화



SEJONG UNIVERSITY

임의 접근을 통한 파일 입출력

- Stream 클래스(파생 클래스 FileStream)의 Position 프로퍼티
  - ✓ 현재 스트림의 읽는 위치 또는 쓰는 위치를 나타냄
  - ✓ WriteByte() 또는 ReadByte() 메소드를 호출하면 자동으로 Position이 1씩 증가
  - ✓ Write() 또는 Read()는 쓰거나 읽은 바이트 수만큼 Position이 증가
- 임의 접근(Random access) 방식
  - ✓ Seek() 메소드를 호출하거나 Position 프로퍼티에 원하는 값 대입
  - ✓ 지정한 위치로 점프해 읽기/쓰기를 위한 준비를 할 수 있음

사용예제:

```
Stream outStream = new FileStream("a.dat", FileMode.Create);
// ...
outStream.Seek(5, SeekOrigin.Current); // 현재 위치에서 5바이트 뒤로 이동
outStream.WriteByte(0x04);
```

임의 접근을 통한 파일 입출력 예제 코드

```
static void Main(string[] args)
{
    Stream outStream = new FileStream("a.dat", FileMode.Create);
    Console.WriteLine($"Position : {outStream.Position}");

    outStream.WriteByte(0x01);
    Console.WriteLine($"Position : {outStream.Position}");

    outStream.WriteByte(0x02);
    Console.WriteLine($"Position : {outStream.Position}");

    outStream.WriteByte(0x03);
    Console.WriteLine($"Position : {outStream.Position}");

    outStream.Seek(5, SeekOrigin.Current); // 현재 위치에서 5바이트 뒤로 이동
    Console.WriteLine($"Position : {outStream.Position}");

    outStream.WriteByte(0x04);
    Console.WriteLine($"Position : {outStream.Position}");

    outStream.Close();
}
```

출력 결과:

```
Position : 0
Position : 1
Position : 2
Position : 3
Position : 8
Position : 9
```

파일 입출력을 위한 using 키워드

- using 키워드는 파일이나 소켓을 비롯한 자원을 다룰 때 사용 가능
  - ✓ 코드 블록의 마지막에 Dispose() 메소드를 호출
  - ✓ Stream.Close() 메소드도 Dispose()를 호출하기 때문에 사실상 동일한 코드

Stream.Close() 호출:

```
long someValue = 0x123456789ABCDEF0;

//  1) 파일 스트림 생성
Stream outStream = new FileStream("a.dat", FileMode.Create);

// 2) someValue(long 형식)를 byte 배열로 변환
byte[] wBytes = BitConverter.GetBytes(someValue);

// 3) 변환한 byte 배열을 파일 스트림을 통해 파일에 기록
outStream.Write(wBytes, 0, wBytes.Length);

// 4) 파일 스트림 닫기
outStream.Close();
```

using 키워드 사용:

```
{
    //  1) 파일 스트림 생성
    using Stream outStream =
            new FileStream("a.dat", FileMode.Create);

    // 2) someValue(long 형식)를 byte 배열로 변환
    byte[] wBytes =
                    BitConverter.GetBytes(someValue);

    // 3) 변환한 byte 배열을 파일 스트림을 통해
          파일에 기록
    outStream.Write(wBytes, 0, wBytes.Length);
} // using 선언을 통해 코드 블록이 끝나면서
     outStream.Dispose() 호출
```

파일 입출력을 위한 using 키워드 예제 코드

```
using FS = System.IO.FileStream; // using 별칭 지시문

static void Main(string[] args)
{
    long someValue = 0x123456789ABCDEF0;
    Console.WriteLine("{0,-13} : 0x{1:X16}", "Original Data", someValue);

    using (Stream outStream = new FS("a.dat", FileMode.Create))
    {
        byte[] wBytes = BitConverter.GetBytes(someValue);
        Console.Write("{0,-13} : ", "Byte array");

        foreach (byte b in wBytes)
            Console.Write("{0:X2} ", b);
        Console.WriteLine();

        outStream.Write(wBytes, 0, wBytes.Length);
    } // using 선언문 아래에 코드 블록을 만드는 방법
```

출력 결과:

```
Original Data : 0x123456789ABCDEF0
Byte array    : F0 DE BC 9A 78 56 34 12
Read Data     : 0x123456789ABCDEF0
```

```
    using Stream inStream = new FS("a.dat", FileMode.Open);
    byte[] rbytes = new byte[8];

    int i = 0;
    while (inStream.Position < inStream.Length)
        rbytes[i++] = (byte)inStream.ReadByte();

    long readValue = BitConverter.ToInt64(rbytes, 0);
    Console.WriteLine("{0,-13} : 0x{1:X16} ",
                                    "Read Data", readValue);
}
```

이진 데이터 처리 : BinaryWriter/BinaryReader

- FileStream 클래스는 파일처리를 위해 byte 또는 byte 배열형식으로 변환
  - ✓ .NET은 FileStream의 불편함을 해소하기 위해 도우미 클래스 제공
- BinaryWriter/BinaryReader 클래스
  - ✓ 스트림에 이진 데이터를 기록하거나 읽어 들이기 위한 기능 수행
  - ✓ Stream의 파생 클래스의 인스턴스가 필요

BinaryWriter 메소드 :

```
BinaryWriter bw = new BinaryWriter(new
FileStream("a.dat", FileMode.Create));

// Write()는 모든 기본 데이터형식에 대해 오버로딩
bw.Write(32);
bw.Write("Good Morning!");
bw.Write(3.14);

bw.Close();
```

BinaryReader 메소드 :

```
BinaryReader br = new BinaryReader(new
FileStream("a.dat", FileMode.Open));

// 읽을 데이터 형식별로 ReadInt32처럼
   Read데이터형식() 메소드 제공
int a = br.ReadInt32();
string b = br.ReadString();
double c = br.ReadDouble();

br.Close();
```

이진 데이터 처리 예제 코드

```
static void Main(string[] args)
{
    using (BinaryWriter bw = new BinaryWriter(new FileStream("a.dat",
FileMode.Create)))
    {
        bw.Write(int.MaxValue);
        bw.Write("Good Morning!");
        bw.Write(uint.MaxValue);
        bw.Write("안녕하세요.");
        bw.Write(double.MaxValue);
    }

    using BinaryReader br = new BinaryReader(new FileStream("a.dat",
FileMode.Open));
    Console.WriteLine($"File size : {br.BaseStream.Length} bytes");
    Console.WriteLine(br.ReadInt32());
    Console.WriteLine(br.ReadString());
    Console.WriteLine(br.ReadUInt32());
    Console.WriteLine(br.ReadString());
    Console.WriteLine(br.ReadDouble());
}
```

출력 결과:

```
File size : 47 bytes
2147483647
Good Morning!
4294967295
안녕하세요.
1.7976931348623157E+308
```

텍스트 파일 처리 : StreamWriter/StreamReader

- 텍스트 파일은 구조는 간단하고 활용도가 높은 파일 형식
  - ✓ ASCII 인코딩에서는 각 바이트가 문자 하나를 나타냄
  - ✓ ASCII 인코딩의 경우, 바이트 오더 문제가 없고 플랫폼에 상관없이 생성하고 읽음
  - ✓ 생성한 파일의 내용을 편집기로 열면 사람이 바로 읽을 수 있음
- StreamWriter/StreamReader 클래스
  - ✓ 텍스트 파일을 쓰고 읽기 위한 기능을 제공
  - ✓ Stream의 파생 클래스의 인스턴스가 필요

StreamWriter 클래스 사용:

```
StreamWriter sw = new StreamWriter(new
        FileStream("a.dat", FileMode.Create));
// Write() 또는 WriteLine()는 모든 기본
   데이터형식에 대해 오버로딩
sw.Write(32);
sw.WriteLine("Good Morning!");
sw.WriteLine(3.14);
sw.Close();
```

StreamReader 클래스 사용:

```
StreamReader sr = new StreamReader(new
          FileStream("a.dat", FileMode.Open));
// EndOfStream 프로퍼티는 스트림의 끝에
   도달하는지 나타냄
while (sr.EndOfStream == false)
    Console.WriteLine(sr.ReadLine());
sr.Close();
```

텍스트 파일 처리 예제 코드

```
static void Main(string[] args)
{
    using (StreamWriter sw = new StreamWriter(new FileStream("a.txt", FileMode.Create)))
    {
        sw.WriteLine(int.MaxValue);
        sw.WriteLine("Good Morning!");
        sw.WriteLine(uint.MaxValue);
        sw.WriteLine("안녕하세요.");
        sw.WriteLine(double.MaxValue);
    }

    using (StreamReader sr = new StreamReader(new FileStream("a.txt", FileMode.Open)))
    {
        Console.WriteLine($"File size : {sr.BaseStream.Length} bytes");
        while (sr.EndOfStream == false)
            Console.WriteLine(sr.ReadLine());
    }
}
```

출력 결과:

```
File size : 82 bytes
2147483647
Good Morning!
4294967295
안녕하세요.
1.79769313486231570E+308
```

객체 직렬화 (Serialization)

- 개발자가 정의한 복합 데이터형식을 쉽게 스트림에 쓰고 읽기
  - ✓ 복합 데이터형식이 가진 필드 값을 저장할 순서를 정하고,
  - ✓ 정해진 순서에 따라 저장하고 읽을 수 있도록 지원
  - ✓ [Serializable] 애트리뷰트를 클래스 선언부 앞에 붙여줌

사용형식:

```
[Serializable]
class MyClass
{
    // ...
}
```

직렬화하여 저장

```
Stream ws = new FileStream("a.dat", FileMode.Create);
BinaryFormatter serializer = new BinaryFormatter();

MyClass obj = new MyClass();
// obj의 필드에 값 저장

serializer.Serialize(ws, obj);
ws.Close();
```

역직렬화하여 복원

```
Stream rs = new FileStream("a.dat", FileMode.Open);
BinaryFormatter deserializer = new BinaryFormatter();

MyClass obj = (MyClass)deserializer.Deserialize(rs);
rs.Close();
```

객체 직렬화 (Serialization)

- 직렬화하고 싶지 않은 필드 또는 직렬화할 수 없는 필드 처리
  - ✓ 해당 필드를 [NonSerialized] 애트리뷰트로 수식
  - ✓ 직렬화할 때도 저장되지 않고, 역직렬화할 때도 복원되지 않음

```
[Serializable]
class MyClass
{
    public int myField1;
    public int myField2;

    [NonSerialized]
    public int myField3; // 직렬화하고 싶지 않은 필드

    public int myField4;

    [NonSerialized]
    public NonserializedClass myField5; // 직렬화할 수 없는 필드
}
```

객체 직렬화 예제 코드

```
static void Main(string[] args)
{
    using (Stream ws = new FileStream("a.dat", FileMode.Create))
    {
        BinaryFormatter serializer = new BinaryFormatter();

        NameCard nc = new NameCard();
        nc.Name = "홍길동";
        nc.Phone = "010-123-4567";
        nc.Age = 33;

        serializer.Serialize(ws, nc);
    }


    using Stream rs = new FileStream("a.dat", FileMode.Open);
    BinaryFormatter deserializer = new BinaryFormatter();

    NameCard nc2;
    nc2 = (NameCard)deserializer.Deserialize(rs);
    Console.WriteLine($"Name:  {nc2.Name}");
    Console.WriteLine($"Phone:  {nc2.Phone}");
    Console.WriteLine($"Age:  {nc2.Age}");
}
```

```
[Serializable]
class NameCard
{
    public string Name;
    [NonSerialized]
    public string Phone;
    public int Age;
}
```

출력 결과:

```
Name:  홍길동
Phone:
Age:  33
```

컬렉션 객체 직렬화 예제 코드

```
static void Main(string[] args)
{
    using (Stream ws = new FileStream("a.dat", FileMode.Create))
    {
        BinaryFormatter serializer = new BinaryFormatter();

        List<NameCard> list = new List<NameCard>();
        list.Add(new NameCard("홍길동", "010-123-4567", 33));
        list.Add(new NameCard("손오공", "010-123-1111", 22));
        list.Add(new NameCard("사오정", "010-123-2222", 26));

        serializer.Serialize(ws, list);
    }


    using Stream rs = new FileStream("a.dat", FileMode.Open);
    BinaryFormatter deserializer = new BinaryFormatter();

    List<NameCard> list2;
    list2 = (List<NameCard>)deserializer.Deserialize(rs);

    foreach(NameCard nc in list2)
        Console.WriteLine($"Name:  {nc.Name}, Phone:  {nc.Phone}, Age:  {nc.Age}");
}
```

```
[Serializable]
class NameCard
{
    public NameCard(string Name, string Phone,
                                                int Age)
    {
        this.Name = Name;
        this.Phone = Phone;
        this.Age = Age;
    }
    public string Name;
    public string Phone;
    public int Age;
}
```

출력 결과:

```
Name:  홍길동, Phone:  010-123-4567, Age:  33
Name:  손오공, Phone:  010-123-1111, Age:  22
Name:  사오정, Phone:  010-123-2222, Age:  26
```

Thank you!